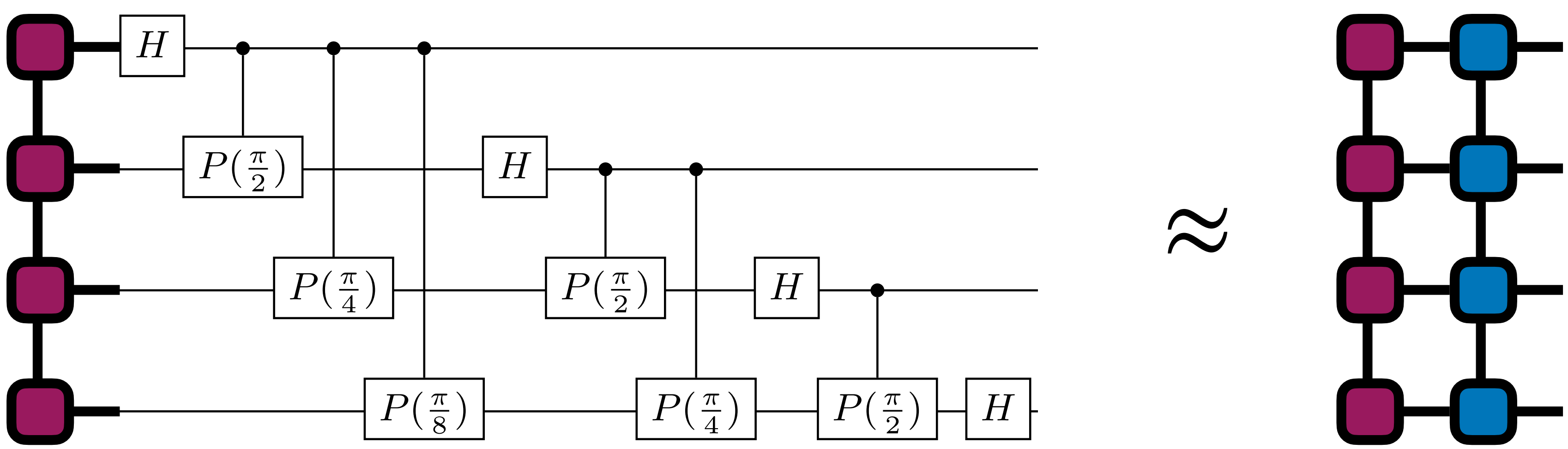
ITensor MPS and MPO Algorithms



E.M. Stoudenmire

FLATIRON INSTITUTE
SIMONS FOUNDATION

August 2024 - U. Mass. Amherst

Matrix Product State (MPS) Algorithms

MPS Algorithms

The ITensorMPS package offers many helpful algorithms for working with MPS and MPO's

- OpSum system – making MPOs from operators
- dmrg & tdvp – computing ground states and dynamics
- expect – computing expected values of operators
- correlation_matrix – compute correlation functions
- inner – overlap MPS and MPOs
- contract, sum – algebra of MPS and MPO

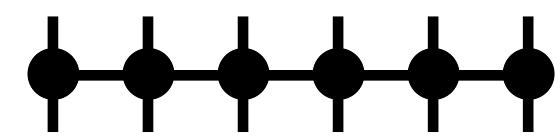
MPS Algorithms

OpsSum – powerful "domain-specific language" (DSL) for making MPOs from math expressions

$$\sum_j S^z_j S^z_{j+1} \longrightarrow$$

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms += "Sz",j, "Sz",j+1
end

H = MPO(terms,sites)
```

$\longrightarrow$



H

MPS Algorithms

Starting from beginning: first make an array of "sites"
Just a Julia array of Index objects

```julia
sites = siteinds("S=1/2",N)
```

MPS Algorithms

Next fill up OpSum with "terms" of the operator

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms += "Sz",j, "Sz",j+1
end
```

Internal data of "terms" similar to:

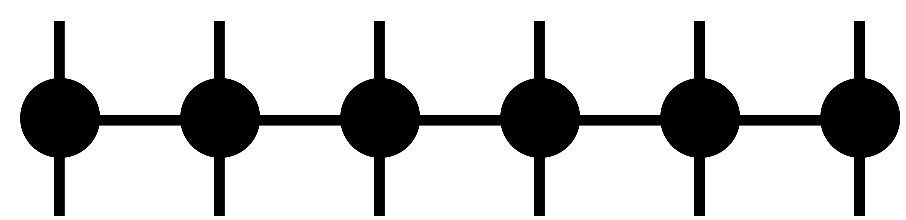
(1.0,"Sz",1,"Sz",2),   (1.0,"Sz",2,"Sz",3), ...

MPS Algorithms

Finally, construct MPO – compresses terms together [1,2]

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms += "Sz",j, "Sz",j+1
end

H = MPO(terms,sites)
```

(1.0,"Sz",1,"Sz",2), (1.0,"Sz",2,"Sz",3), ...

$H =$ 

Often optimal "bond dimension" reached

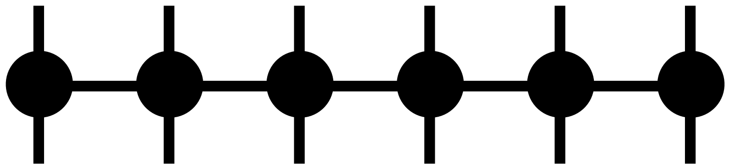
[1] Chan, Keselman, Nakatani, Li, White, J. Chem. Phys 145 (2016)
[2] Parker, Zaletel, Phys. Rev. B, 102, 035147 (2020)

MPS Algorithms

Very wide range of operators can be made

$$H = \sum_j S^z_j S^z_{j+1}$$

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms += "Sz",j, "Sz",j+1
end

H = MPO(terms,sites)
```

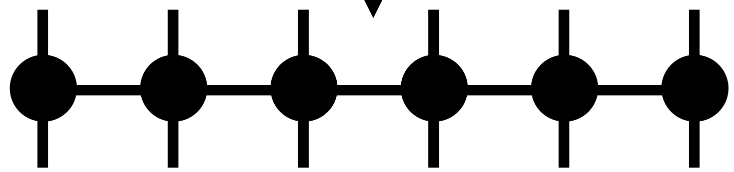
$H =$ 

MPS Algorithms

Very wide range of operators can be made

$$H = \sum_j S^z_j S^z_{j+1} + \frac{1}{2} S^+_j S^-_{j+1} + \frac{1}{2} S^-_j S^+_{j+1}$$

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms +=       "Sz",j, "Sz",j+1
  terms += 1/2,"S+",j, "S-",j+1
  terms += 1/2,"S-",j, "S+",j+1
end

H = MPO(terms,sites)
```

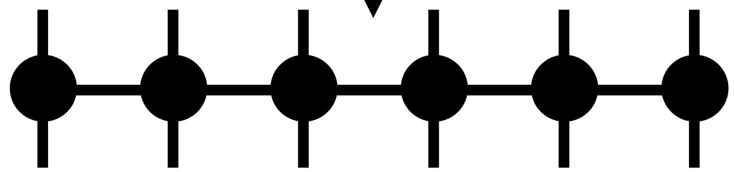
$H =$ 

MPS Algorithms

Changing site type automatically gives correct operators

$$H = \sum_j S^z_j S^z_{j+1} + \frac{1}{2} S^+_j S^-_{j+1} + \frac{1}{2} S^-_j S^+_{j+1}$$

```
sites = siteinds("S=1/2",N)

terms = OpSum()
for j=1:N-1
  terms +=      "Sz",j, "Sz",j+1
  terms += 1/2,"S+",j, "S-",j+1
  terms += 1/2,"S-",j, "S+",j+1
end

H = MPO(terms,sites)
```

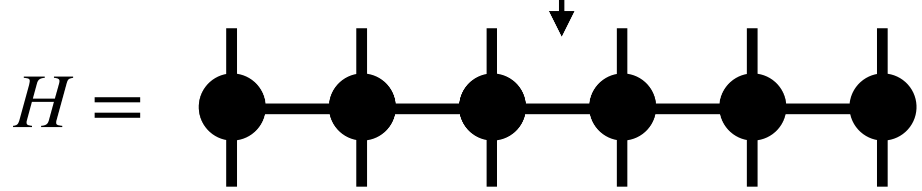
$H =$ 

MPS Algorithms

Changing site type automatically gives correct operators

$$H = \sum_j S^z_j S^z_{j+1} + \frac{1}{2} S^+_j S^-_{j+1} + \frac{1}{2} S^-_j S^+_{j+1}$$

```
sites = siteinds("S=1",N)

terms = OpSum()
for j=1:N-1
  terms +=       "Sz",j, "Sz",j+1
  terms += 1/2,"S+",j, "S-",j+1
  terms += 1/2,"S-",j, "S+",j+1
end

H = MPO(terms,sites)
```

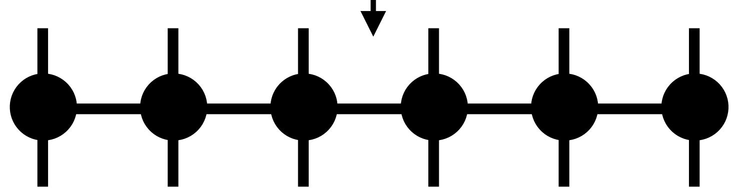


MPS Algorithms

Particles (bosons, fermions) possible too

$$H = \sum_j c_j^\dagger c_{j+1} + c_{j+1}^\dagger c_j$$

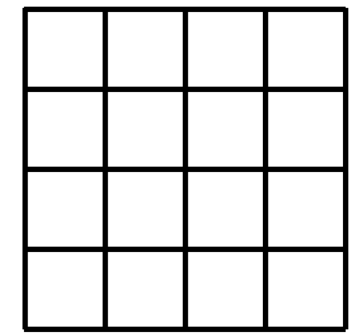
```
sites = siteinds("Fermion",N)

terms = OpSum()
for j=1:N-1
  terms += "Cdag",j, "C",j+1
  terms += "C",j+1, "Cdag",j+1
end

H = MPO(terms,sites)
```

$H =$ 

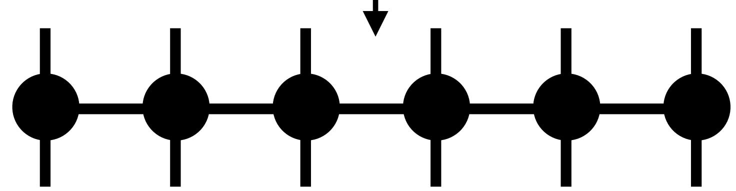
MPS Algorithms

And quasi-two-dimensional systems

$$H = \sum_{\langle ij \rangle} S^z_i S^z_j + \frac{1}{2} S^+_i S^-_j + \frac{1}{2} S^-_i S^+_j$$



```
lattice = square_lattice(Nx, Ny; yperiodic=false)

terms = OpSum()
for b in lattice
  terms +=      "Sz", b.s1, "Sz", b.s2
  terms += 1/2, "S+", b.s1, "S-", b.s2
  terms += 1/2, "S-", b.s1, "S+", b.s2
end

H = MPO(terms, sites)
```

$H =$ 

MPS Algorithms

Next let's look at dmrg and tdvp

- [x] OpSum system – making MPOs from operators
- [ ] dmrg & tdvp – computing ground states and dynamics
- [ ] expect – computing expected values of operators
- [ ] correlation_matrix – compute correlation functions
- [ ] inner – overlap MPS and MPOs
- [ ] contract, sum – algebra of MPS and MPO

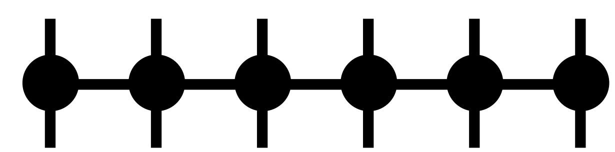
MPS Algorithms

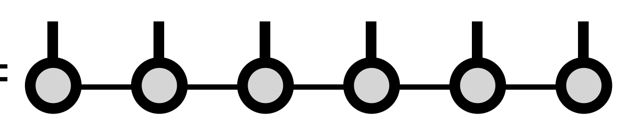
ITensorMPS offers "black box" DMRG algorithm

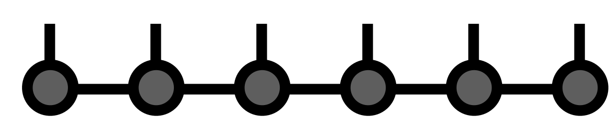
```julia
using ITensors, ITensorMPS

N = 100
sites = siteinds("S=1", N)

terms = OpSum()
for j in 1:(N - 1)
  terms += "Sz", j, "Sz", j + 1
  terms += 0.5, "S+", j, "S-", j + 1
  terms += 0.5, "S-", j, "S+", j + 1
end
H = MPO(terms, sites)

psi0 = random_mps(sites; linkdims=10)

nsweeps = 5
maxdim = [10, 20, 100, 100, 200]
cutoff = [1E-11]
energy, psi = dmrg(H, psi0; nsweeps, maxdim, cutoff)
```

$$H = \sum_j S^z_j S^z_{j+1} + \frac{1}{2} S^+_j S^-_{j+1} + \frac{1}{2} S^-_j S^+_{j+1}$$

(1.0,"Sz",1,"Sz",2), (1.0,"Sz",2,"Sz",3), ...

$H =$ 

$\psi_0 =$ 

$\psi =$ 

MPS Algorithms

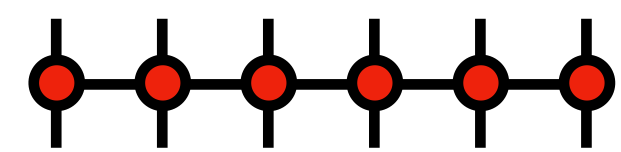
TDVP is similar, for <u>time evolution</u> by some H

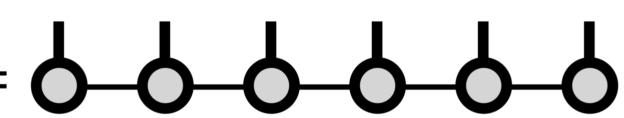
```julia
using ITensors, ITensorMPS

N = 100
sites = siteinds("S=1/2", N)

terms = OpSum()
for j in 1:(N - 1)
  terms += "Sz", j, "Sz", j + 1
  terms += 0.5, "S+", j, "S-", j + 1
  terms += 0.5, "S-", j, "S+", j + 1
end
H = MPO(terms, sites)

psi0 = random_mps(sites; linkdims=10)

t = 10
time_step = 0.1
maxdim = 200
cutoff = 1E-10
psi_t = tdvp(H, -im*t, psi0; maxdim, cutoff, time_step)
```

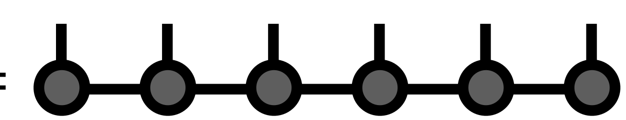
👀

$$H = \sum_j S^z_j S^z_{j+1} + \frac{1}{2} S^+_j S^-_{j+1} + \frac{1}{2} S^-_j S^+_{j+1}$$

(1.0,"Sz",1,"Sz",2), (1.0,"Sz",2,"Sz",3), ...

$e^{-iHt} =$ 

$\psi(0) =$ 

$\psi(t) =$ 

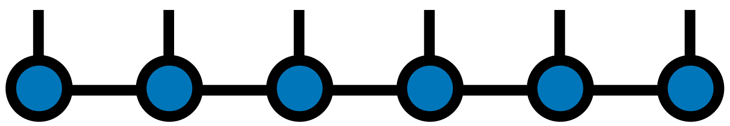
MPS Algorithms

Can use expect, correlation_matrix, and inner to analyze MPS

- ☑ OpSum system – making MPOs from operators
- ☑ dmrg & tdvp – computing ground states and dynamics
- ☐ expect – computing expected values of operators
- ☐ correlation_matrix – compute correlation functions
- ☐ inner – overlap MPS and MPOs
- ☐ contract, sum – algebra of MPS and MPO

MPS Algorithms

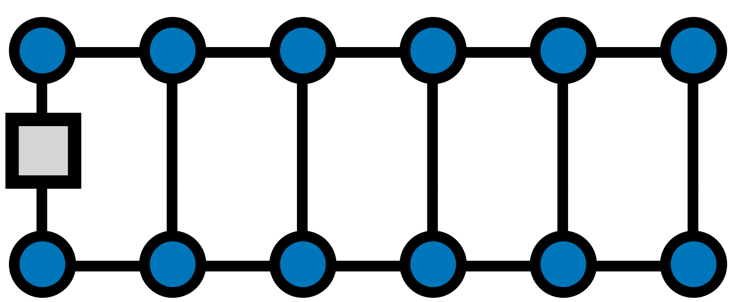
If we have an MPS and want expected values of local operators, can use expect function

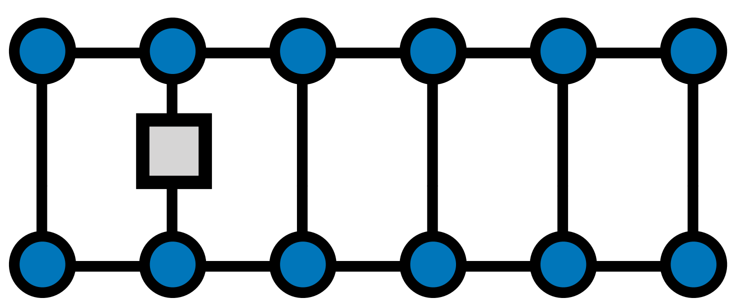
$\Psi =$  could be output of DMRG, TDVP

Then, for example, calling expect gives

```
sz = expect(psi,"Sz")
```

where sz is an array such that

sz[1] = 

sz[2] = 

etc.

MPS Algorithms

Of course, can use various operators for spins, particles, or qubit sites

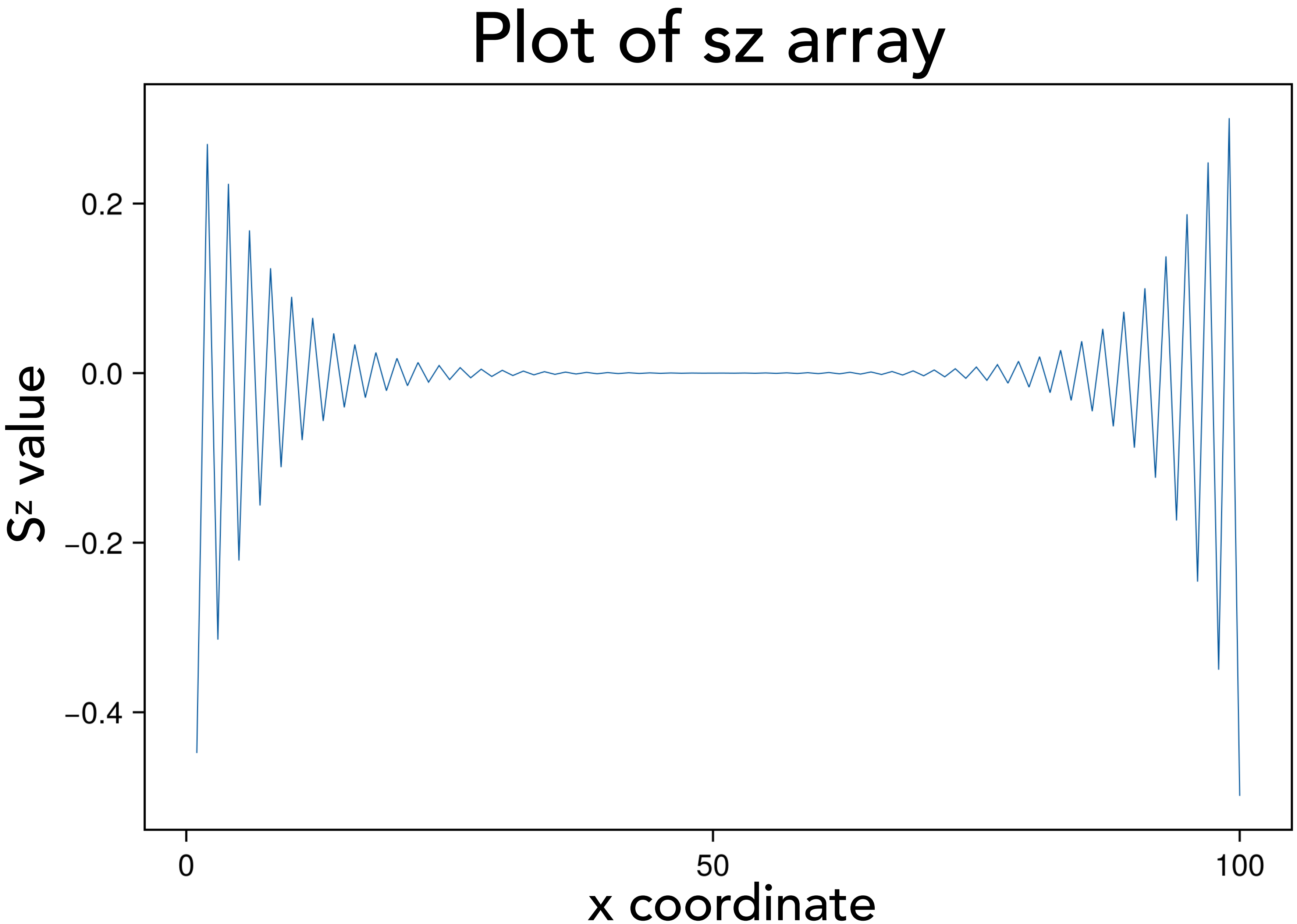
Some examples:

`sz = expect(psi,"Sz")`	magnetization of spins
`density = expect(psi,"N")`	density of particles
`Xvals = expect(psi,"X")`	<X> over a set of qubits

MPS Algorithms

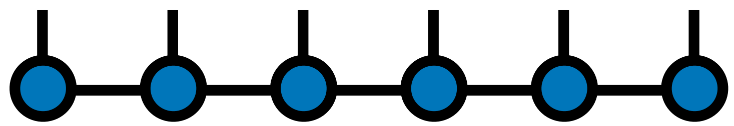
Example output:

```
energy, psi = dmrg(H, psi0; nsweeps, maxdim, cutoff)
sz = expect(psi,"Sz")
```



MPS Algorithms

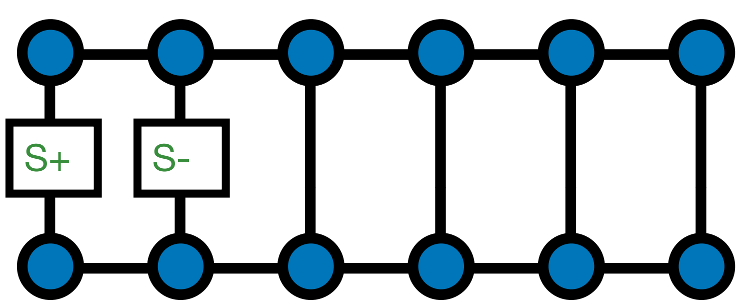
The correlation_matrix function also computes expected values but of "two point" functions i.e. correlators

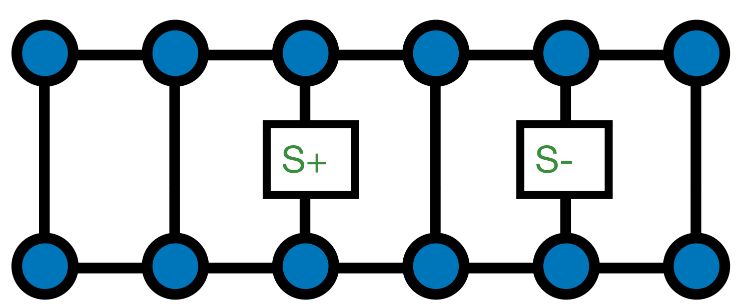
$\Psi =$  could be output of DMRG, TDVP

Then, calling correlation_matrix gives

```
spm = correlation_matrix(psi,"S+","S-")
```

where spm is an Matrix such that

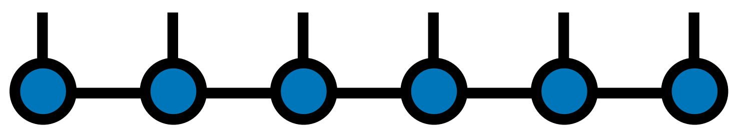
spm[1,2] = 

spm[3,5] = 

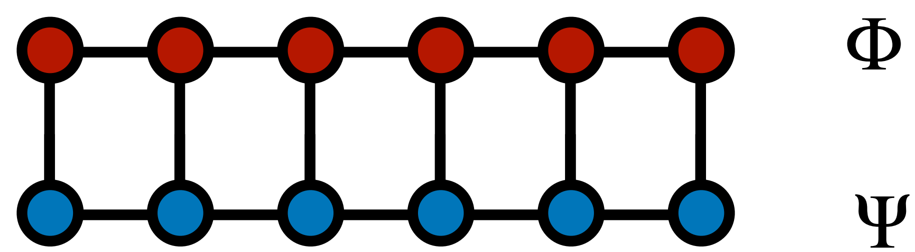
etc.

MPS Algorithms

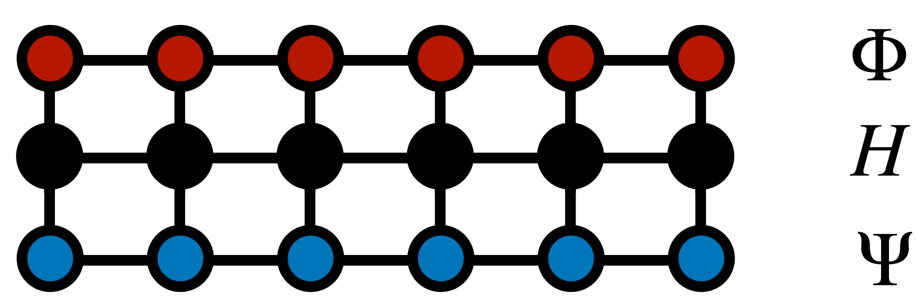
The inner function lets us analyze MPS through overlaps with other MPS and MPOs

$\Psi =$  could be output of DMRG, TDVP

Then, calling inner with another MPS gives

inner(psi,phi) = 

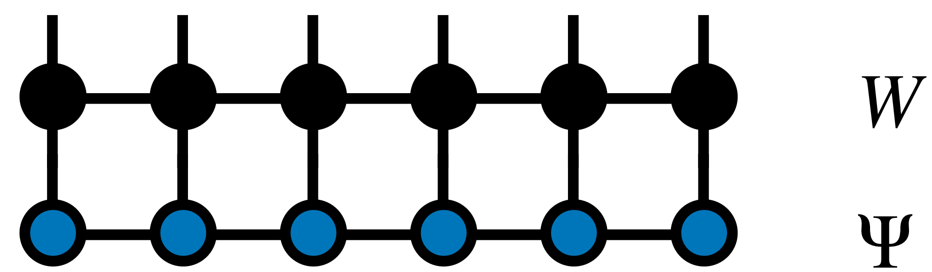
Or including an MPO like

inner(psi',H,phi) = 

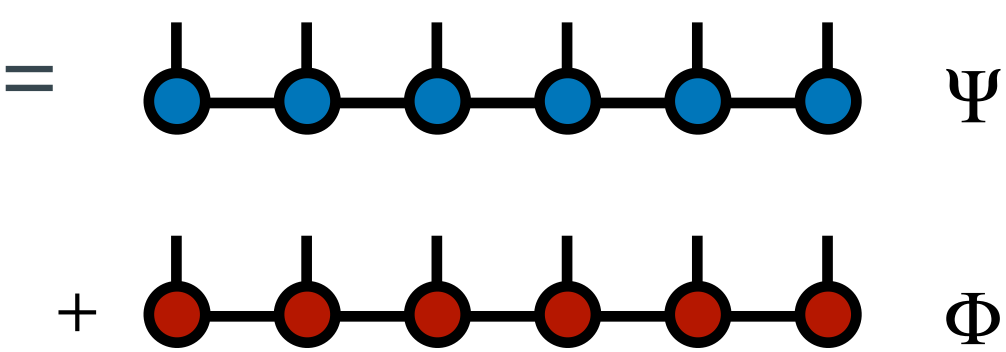
MPS Algorithms

Finally MPS and MPO tensor networks can be contracted and added with contract and sum

contract(W,psi; cutoff) =



sum(psi,phi; cutoff) =



Tensor Factorizations

Review: *Singular Value Decomposition (SVD)*

Given rectangular (4x3) matrix M

$$M = \begin{bmatrix} 0.435839 & 0.223707 & 0.10 \\ 0.435839 & 0.223707 & \\ -0.10 & & \\ 0.223707 & 0.435839 & 0.10 \end{bmatrix}$$

Can factorize as

$$\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & \end{bmatrix} \begin{bmatrix} 0.707107 & 0.707107 \\ 0 & \\ -0.707107 & 0.707107 \end{bmatrix}$$

$$
\underbrace{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}}_{U}
\underbrace{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & \end{bmatrix}}_{S}
\underbrace{\begin{bmatrix} 0.707107 & 0.707107 \\ 0 & \\ -0.707107 & 0.707107 \end{bmatrix}}_{V^T}
$$

Matrices U and V have orthonormal columns:

$$
U^T U = 1
$$

$$
V^T V = 1
$$

S diagonal = "singular values"

Elements of S always:

1) Real
2) Non-negative
3) Decreasing

Keep fewer and fewer elements of S:

$$\overset{U}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}} \overset{S}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & \end{bmatrix}} \overset{V^T}{\begin{bmatrix} 0.707107 & 0.707107 \\ 0 & \\ -0.707107 & 0.707107 \end{bmatrix}}$$

$$= M = \begin{bmatrix} 0.435839 & 0.223707 & 0.10 \\ 0.435839 & 0.223707 & \\ -0.10 & & \\ 0.223707 & 0.435839 & 0.10 \end{bmatrix}$$

$$||M - M||^2 = 0$$

Keep fewer and fewer elements of S:

$$
\overset{\color{red}{U}}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}}
\overset{S}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & 0 \end{bmatrix}}
\overset{\color{blue}{V^T}}{\begin{bmatrix} 0.707107 & 0.707107 \\ 0 & \\ -0.707107 & 0.707107 \end{bmatrix}}
$$

$$
= M_2 = \begin{bmatrix} 0.435839 & 0.223707 & 0 \\ 0.435839 & 0.223707 & 0 \\ 0.223707 & 0.435839 & 0 \\ 0.223707 & 0.435839 & 0 \end{bmatrix}
$$

$$
||M_2 - M||^2 = 0.04 = (0.2)^2
$$

Keep fewer and fewer elements of S:

$$
\overset{\color{red}{U}}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}}
\overset{S}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}
\overset{\color{blue}{V^T}}{\begin{bmatrix} 0.707107 & 0.707107 \\ 0 & \\ -0.707107 & 0.707107 \end{bmatrix}}
$$

$$
= M_3 = \begin{bmatrix} 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \end{bmatrix}
$$

Truncating SVD =

Controlled approximation for M

$$
||M_3 - M||^2 = 0.13 = (0.3)^2 + (0.2)^2
$$

Keep fewer and fewer elements of S:

$$\underset{U}{\begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}} \quad \underset{S}{\begin{bmatrix} 0.933 \end{bmatrix}} \quad \underset{V^T}{\begin{bmatrix} 0.707107 & 0.707107 \end{bmatrix}}$$

$$= M_3 = \begin{bmatrix} 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \end{bmatrix}$$

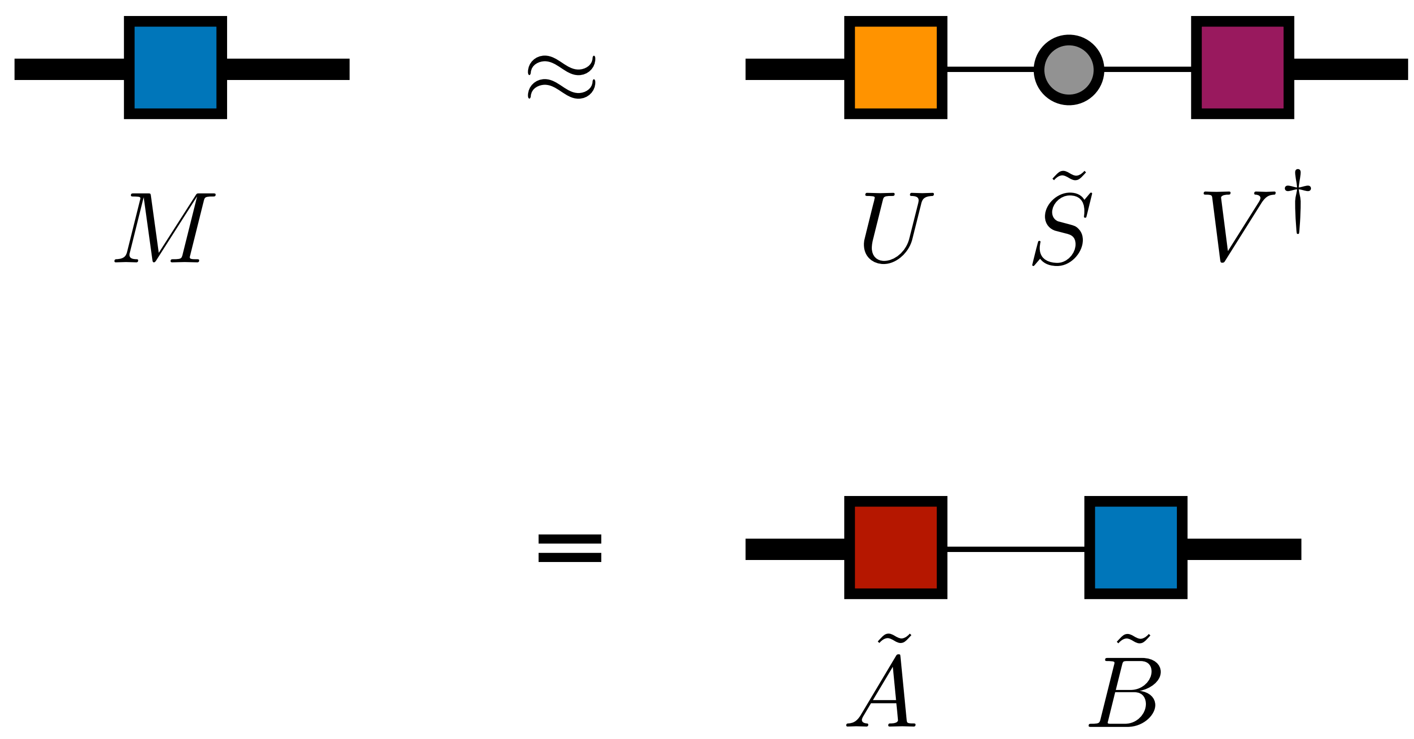
Truncating SVD =

Controlled approximation for M

$$||M_3 - M||^2 = 0.13 = (0.3)^2 + (0.2)^2$$

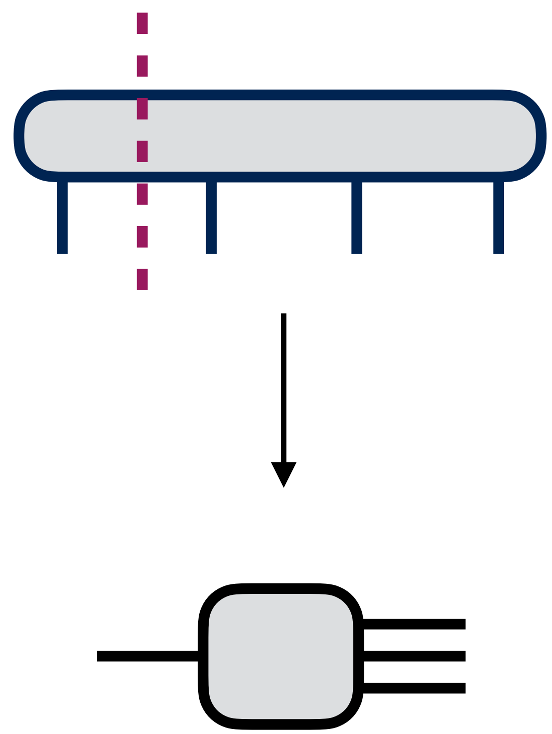
Low-rank Structure

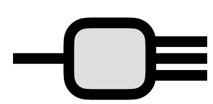
If matrix M approximately low-rank,
truncating singular values of SVD gives optimal approximation

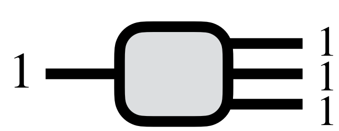
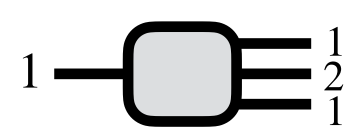


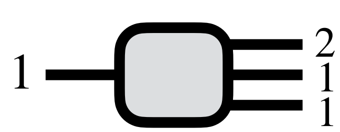
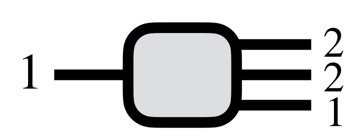
Let's apply SVD to a tensor - how?

Reshape as a matrix:



Reshaping  as a matrix means treating as a 2x8 matrix M, where:

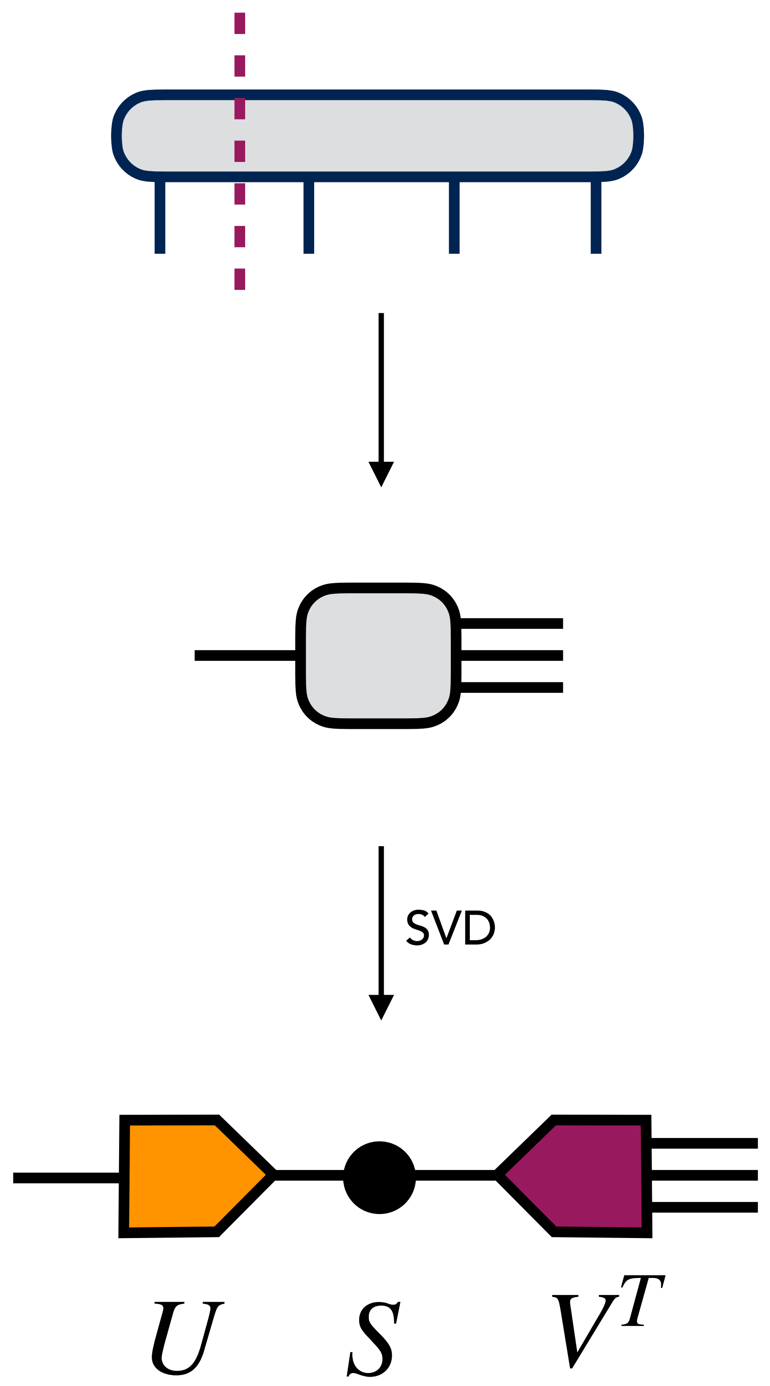
 $= M_{11}$  $= M_{13}$

 $= M_{12}$  $= M_{14}$

etc.

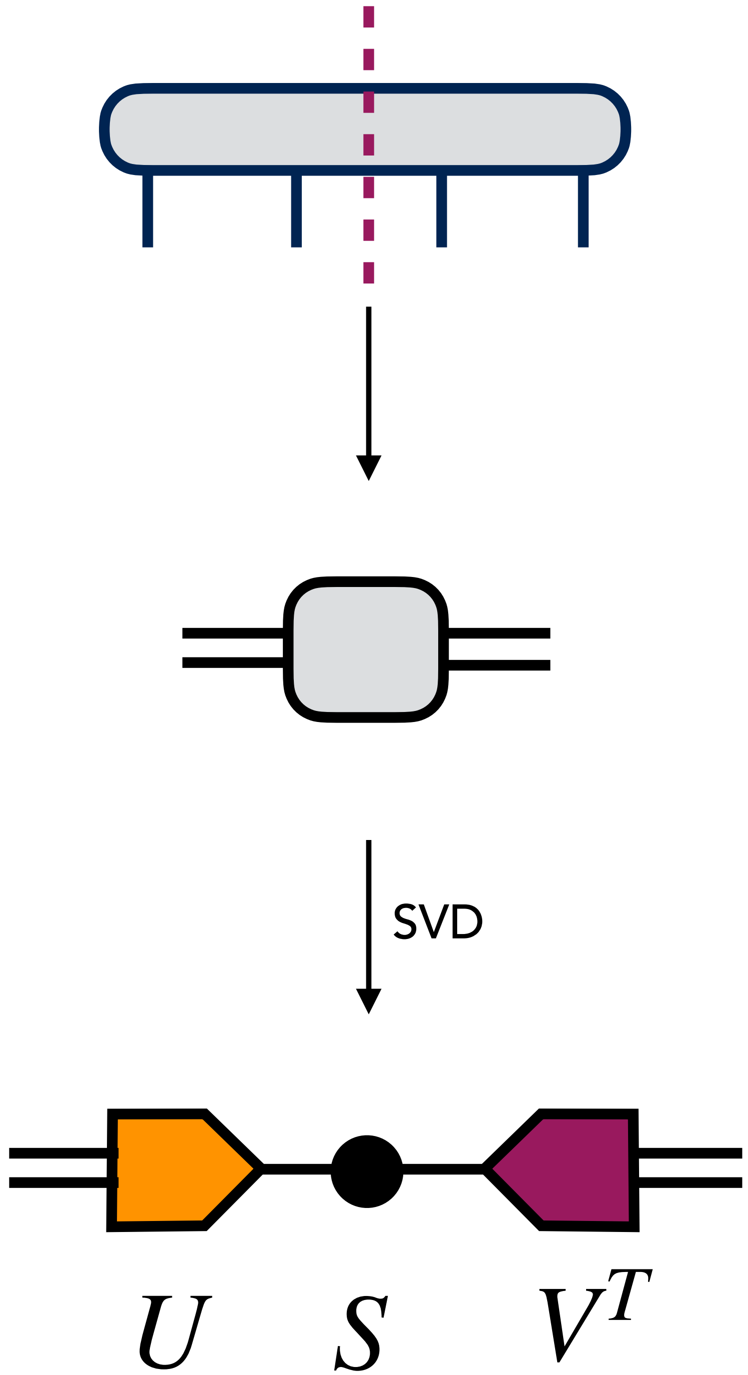
How to generalize SVD to tensors?

Reshape as a matrix:



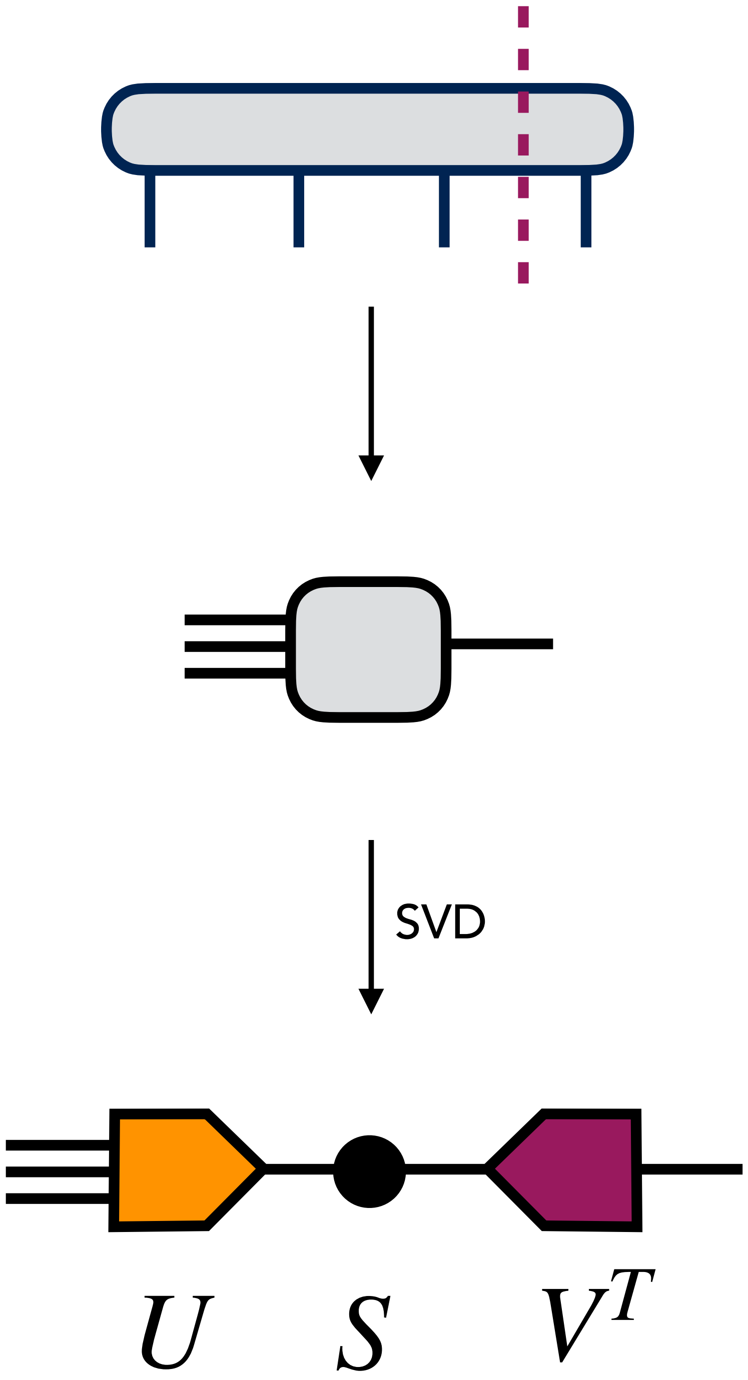
How to generalize SVD to tensors?

Other partitions:

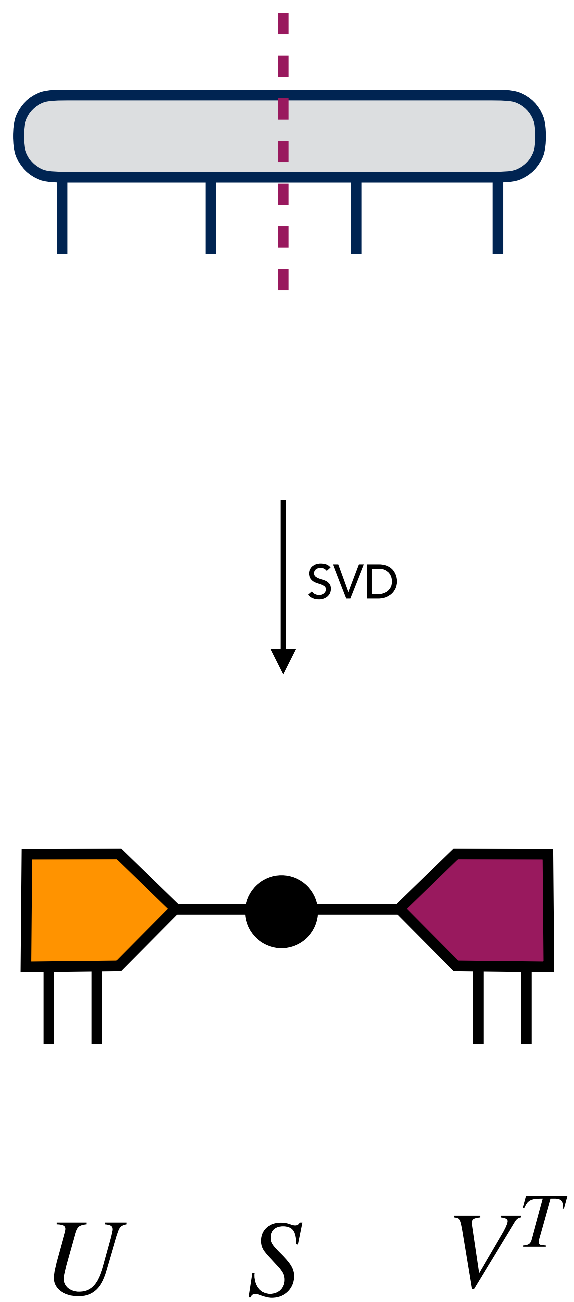


How to generalize SVD to tensors?

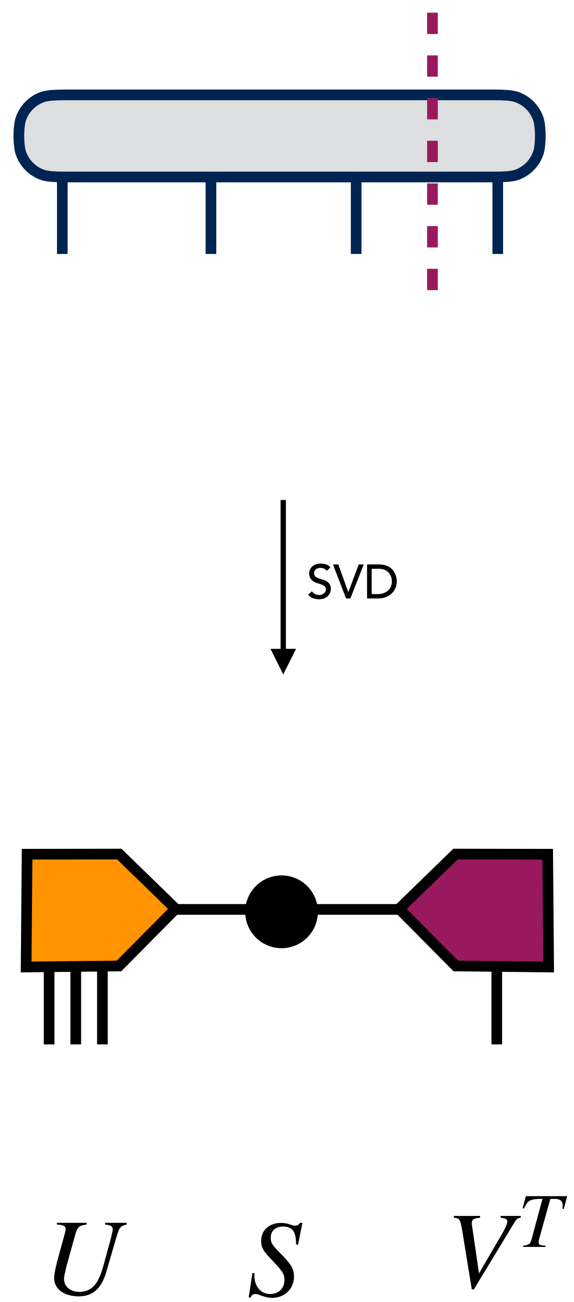
Other partitions:



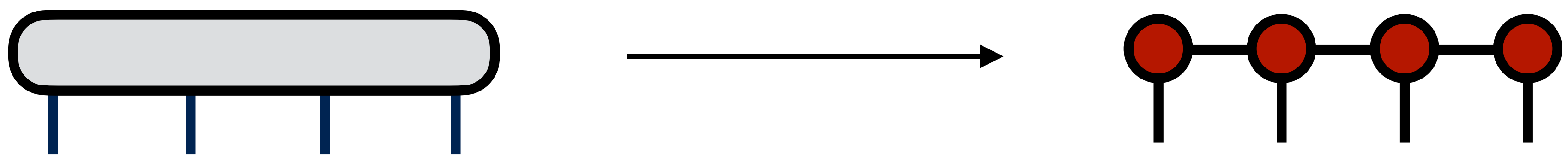
From now on, reshaping steps are *implicit*:



From now on, reshaping steps are *implicit*:

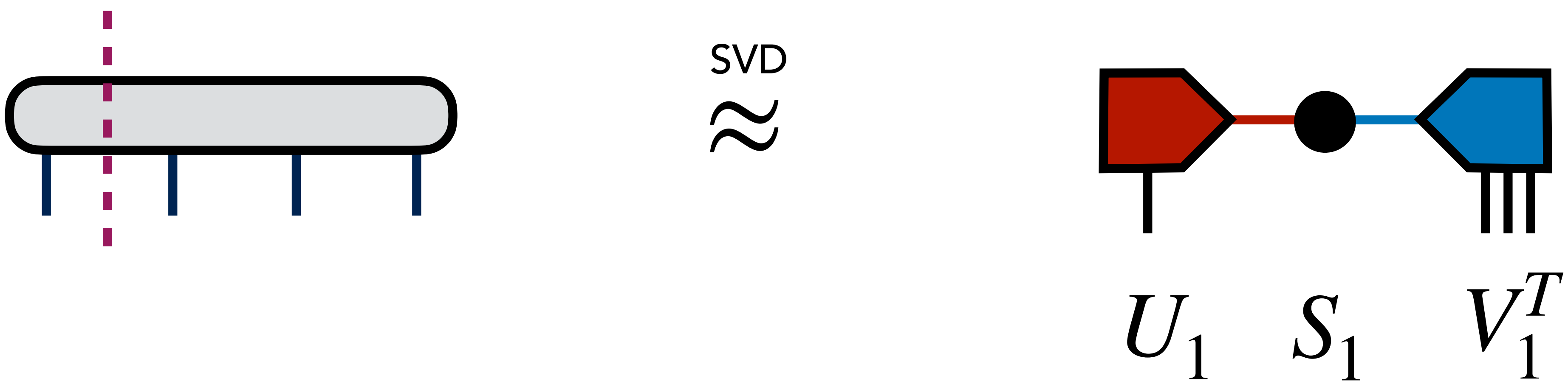


How to compress into a matrix product state?

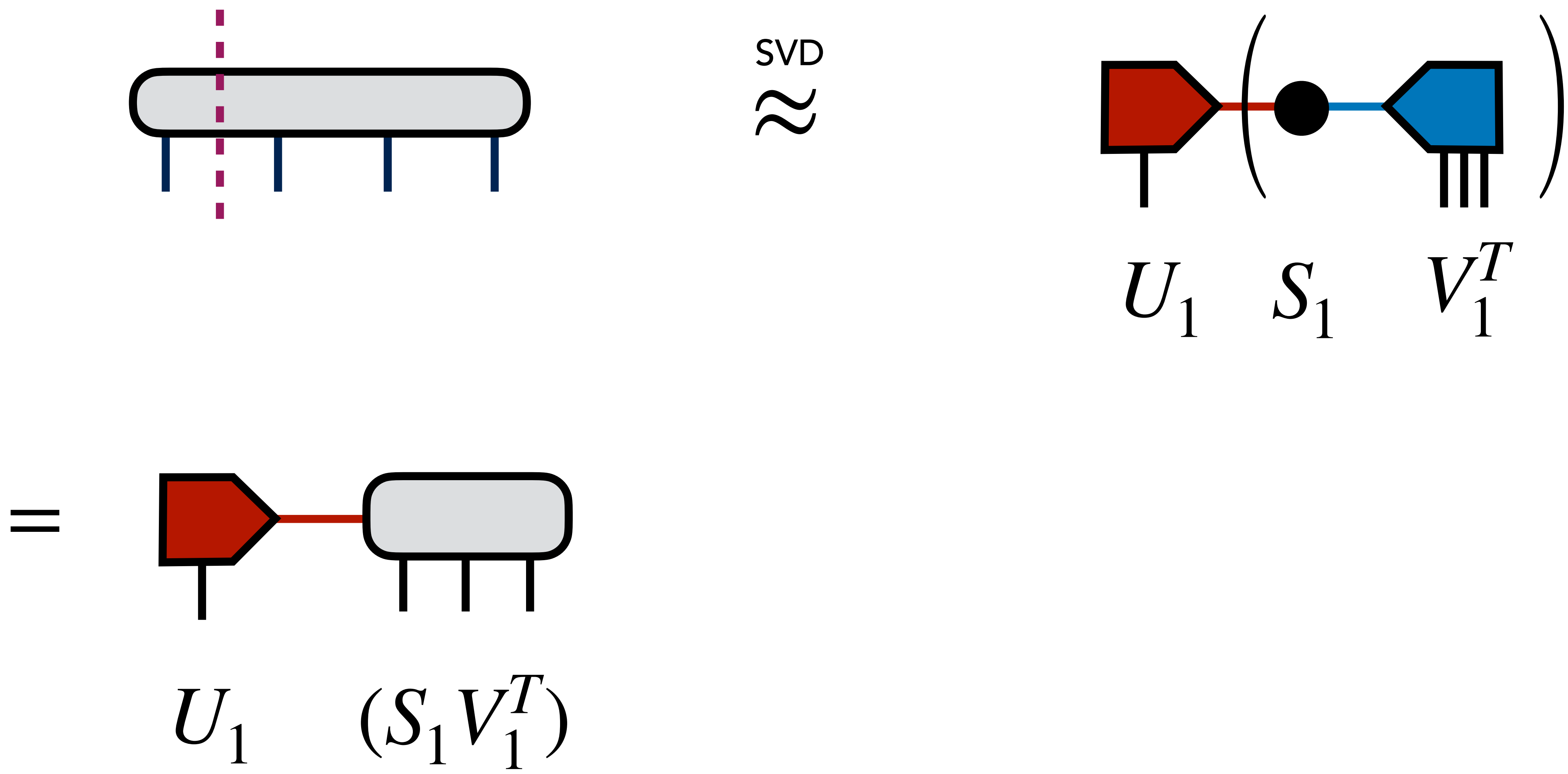


Proceed by sequence of SVD's...

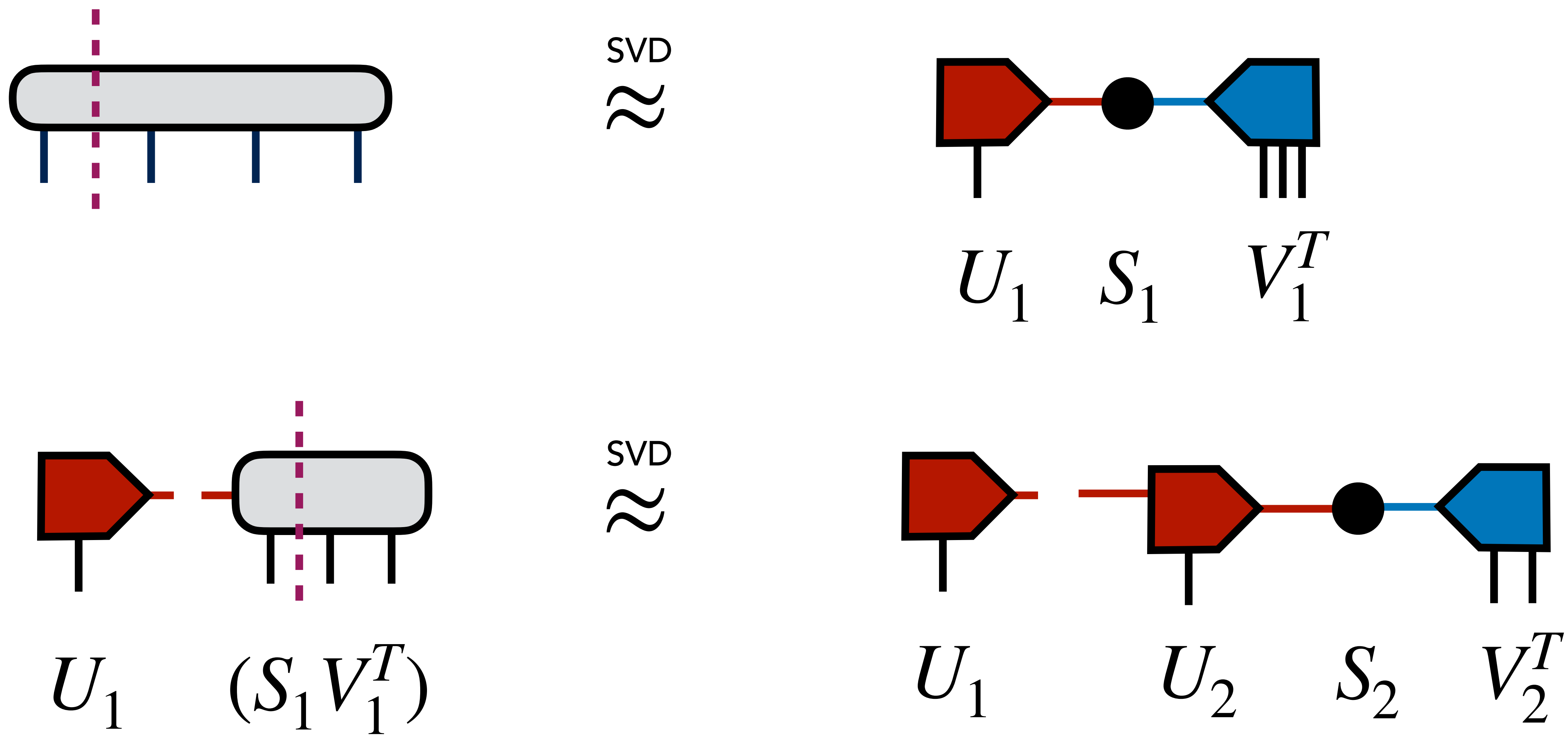
1. SVD first index from rest



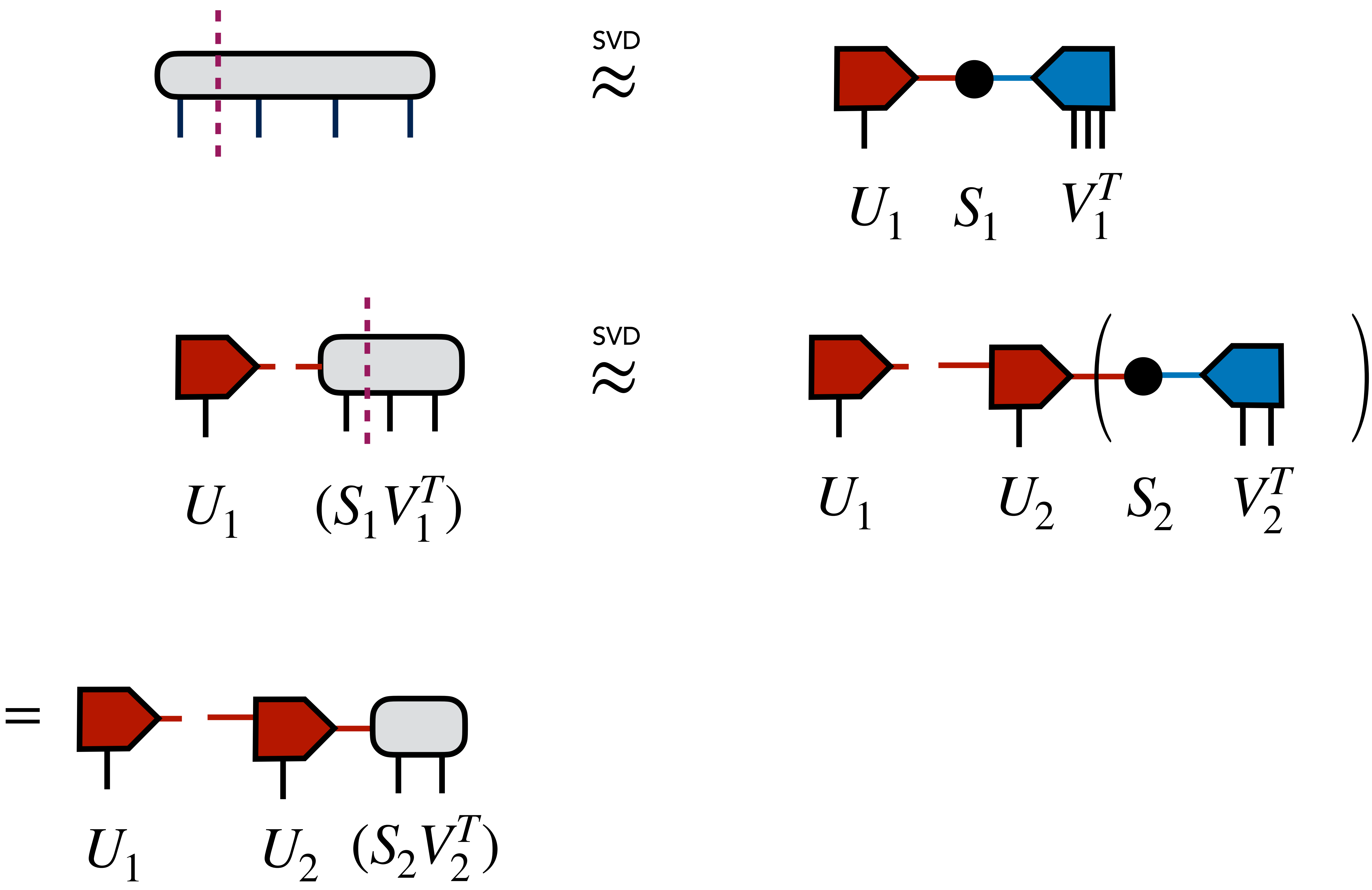
2. Multiply S_1 into V_1^T



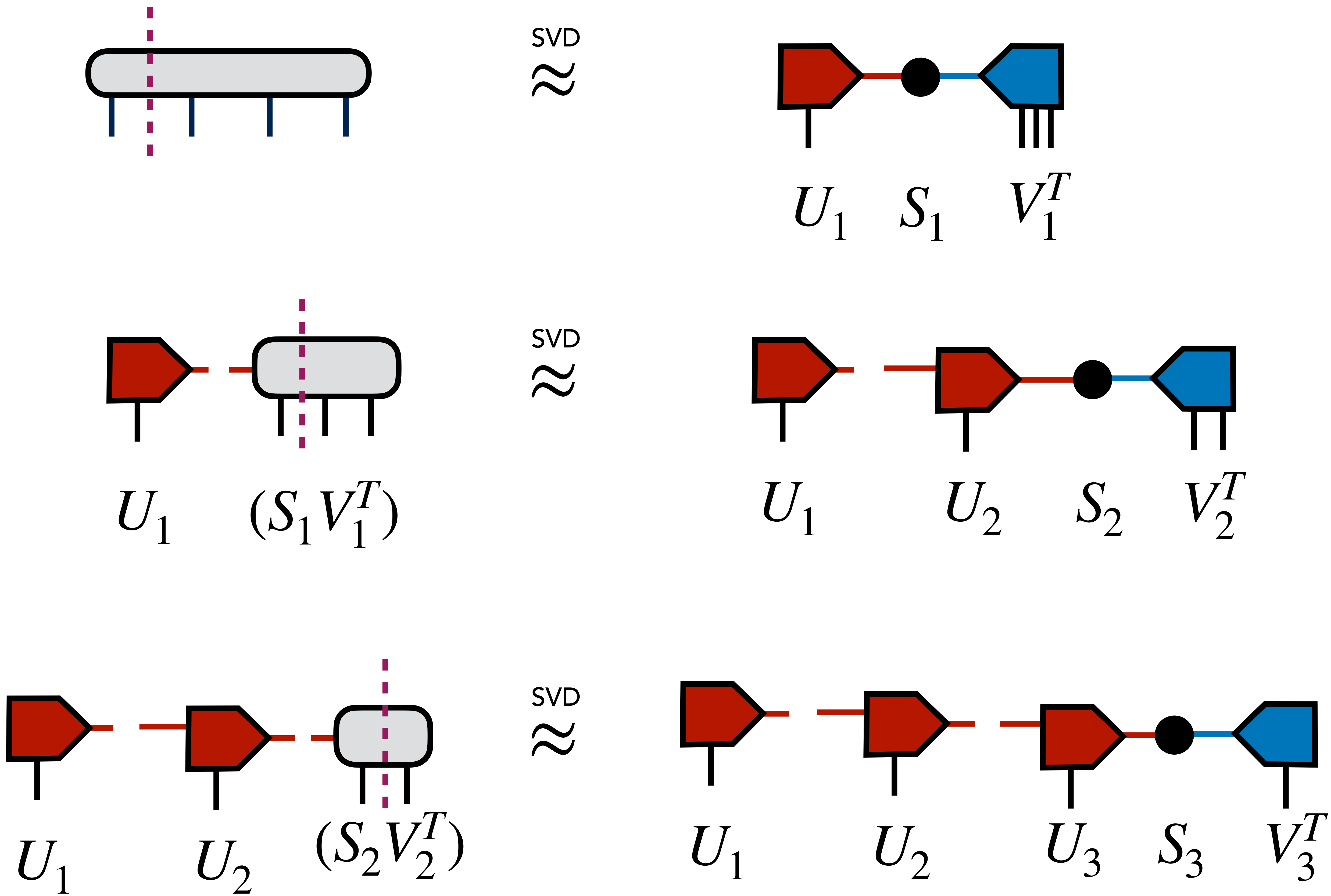
3. SVD this new tensor $\quad (= S_1 V_1^T)$



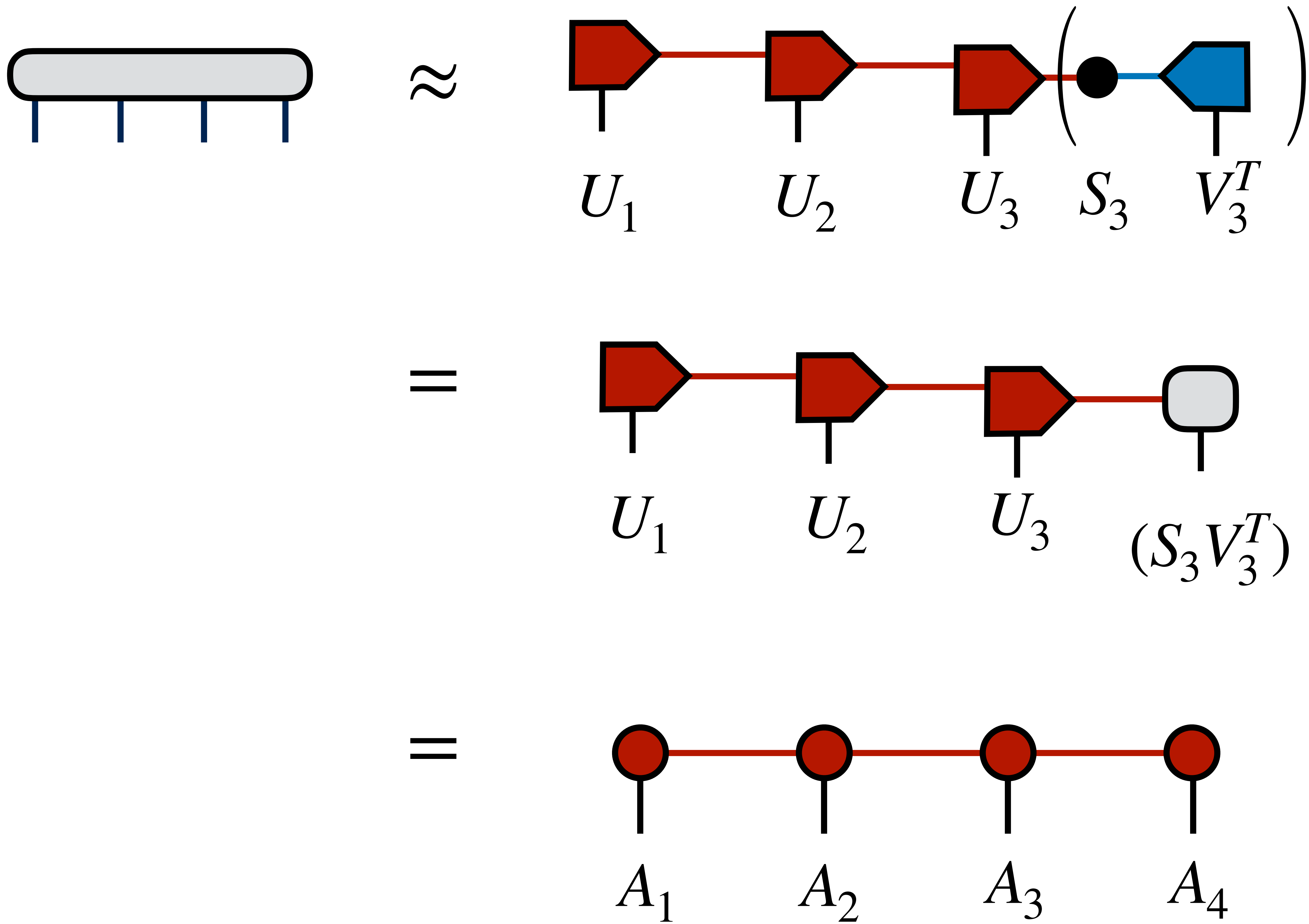
4. Multiply S_2 into V_2



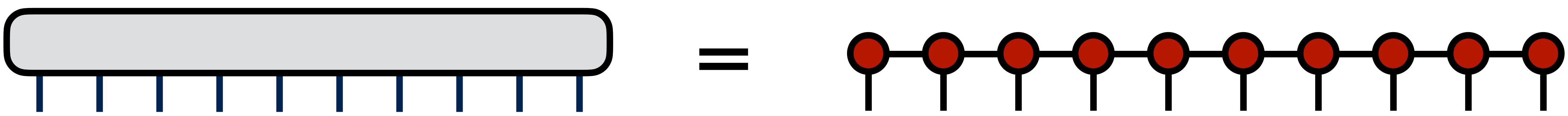
5. Finally SVD $(S_2V_2^T)$



6. Interpret result as an MPS



Matrix product state (MPS) tensor network



Can view as multi-SVD of a tensor

Or special class or subspace of tensors
(low-rank subspace)

Computing Tensor Factorizations in ITensor